APPENDIX I - Playback Engine Partial Exemplary Code

Although aspects of the invention have been described in considerable detail, Appendix I provides a sample of exemplary code so that some additional insight may be gained as to its structure and operation.

```
/*
These are example functions from a Story playback engine which illustrate one possible software implementation of a remarkably lightweight Story operating environment.

These functions illustrate most all the functionality needed for the story multi-threading, media synchronization and runtime model for Story playback.

The first two functions perform the functions of implementing a round-robin, multi-threaded operating system.

The second two functions illustrate functions that implement actual Story op-code execution.
*/

/*
StoryPlaybackCycle should be called continually in a loop on a single host operating system thread.

This functions executes all the threads once in order, until each thread gives up control, then returns.

Possible return code #defines can be found in pStory.h and end with the suffix, "_RETURN_CODE"

When the return value is negative, then execution of the calling loop should end.
*/
S32 FUNC_PREFIX StoryPlaybackCycle(void)
{
  SU32 u32_NumberOfActiveThreads=0;

  SU32 u32_NumberOfThreadsLeft=p.c.u32_NumberOfInitializedThreads; /*
number of initialized threads */
  p.c.u32_StoryPlaybackCycleNumber++;
  p.c.u32_StoryThreadIndex=0;
  while (u32_NumberOfThreadsLeft)
  {
    p.c.context=p.c.contexts[p.c.u32_StoryThreadIndex++];

    if (p.c.context.u32_State!=RUNNING_CONTEXT_STATE)
    {

u32_NumberOfThreadsLeft-=(p.c.context.u32_State!=UNINITIALIZED_CONTEXT_STATE
);
      continue; /* this thread is not running so do next thread */
    }
    u32_NumberOfActiveThreads++;

    if (InputAvailable())
    {
      do
      {
        ProcessInstruction();
      } while
```

```
(p.c.s32_ProcessInstructionReturnCode==SUCCESS_RETURN_CODE);
        if (p.c.s32_ProcessInstructionReturnCode<0)
        {
          break;
        }
      }

      p.c.contexts[p.c.u32_StoryThreadIndex-1]=p.c.context;
      u32_NumberOfThreadsLeft--;
    }
    if (u32_NumberOfActiveThreads==0)
    {
      p.c.s32_ProcessInstructionReturnCode=NO_ACTIVE_THREADS_RETURN_CODE;
    }
    return(p.c.s32_ProcessInstructionReturnCode);
}

/*
This function fetches an opcode from the input buffer and calls the function that implements the
opcode.  It also handles instruction retry by:

Setting the default status returned from the opcode function to
SUCCESS_RETURN_CODE
Storing the pointer to the opcode
Calling the function for the opcode
Inspecting the return code when the opcode function returns
If the return code is RETRY_INSTRUCTION_RETURN_CODE then the instruction pointer is reset to
point back to the opcode by restoring the saved value.

*/
void FUNC_PREFIX ProcessInstruction(void)
{
  PSU32 pu32_SavedNextInput;
  pu32_SavedNextInput=p.c.context.inputBufferInfo.pu32_NextInput;
  p.c.u32_CurrentOpcode=GetSU32_FromInput();
  p.c.s32_ProcessInstructionReturnCode=SUCCESS_RETURN_CODE;
  (controlFunctionAddressArray[p.c.u32_CurrentOpcode])();
  if (p.c.s32_ProcessInstructionReturnCode==RETRY_INSTRUCTION_RETURN_CODE)
  {
    //Instruction could not proceed, so try again next time
    p.c.context.inputBufferInfo.pu32_NextInput=pu32_SavedNextInput;
  }
  return;
}


/*
Stop execution of this thread until all the other threads have had a chance to run.  The return code,
YIELD_TO_NEXT_THREAD_RETURN_CODE, has a different value than a
SUCCESS_RETURN_CODE.

This will cause the main cycle function to move on to executing the next thread.
When the cycle function gets back to executing this thread, execution will proceed starting with the
instruction following the YIELD_OP instruction.
*/
void FUNC_PREFIX YieldOp(void)
{
  p.c.s32_ProcessInstructionReturnCode=YIELD_TO_NEXT_THREAD_RETURN_CODE;
  return;
}
```

```
/*
End ops are used to end subroutines and disable threads.

Note that after the last running thread ends, then the story playback will automatically end.
*/
void FUNC_PREFIX EndOp(void)
{
  RETURN_ADDRESS_STACK_ELEMENT_TYPE rase;
  SU32 u32_i;
  if (p.c.context.u32_SubroutineNestingLevel)
  {
    p.c.context.u32_SubroutineNestingLevel–;
    Pop((PSU8)&rase, sizeof(rase));
    p.c.context.inputBufferInfo=rase.inputBufferInfo;
    p.c.context.pu32_Parameters=rase.pu32_Parameters;
    p.c.context.pFileInfo=rase.pInputFileInfo;
    for
(u32_i=0;u32_i<rase.u32_NumberOfElementsOnStackToPopUponReturn;u32_i++)
    {
      Pop(NULL,0);
    }
  }
  else
  { /* Thread Ended its own Execution */
    p.c.context.u32_State=SUSPENDED_CONTEXT_STATE;

p.c.s32_ProcessInstructionReturnCode=YIELD_TO_NEXT_THREAD_RETURN_CODE;
  }
  return;
}
```

END OF APPENDIX I